Java Telecommunication Application Server Technology Comparison

by Michael Maretzke

29th July 2008

Set The Scope

Yet another religious technology battlefield? The comparison of JAIN SLEE[1] (JSR-22[2] + JSR-240[3]) and SIP Servlets (JSR-116[4] + JSR-289[5]) based application servers reminds sometimes to the never ending debates about which is the "best" operating system, which the "best" browser.
Nevertheless, the discussion has relevance and hence people ask for guidance when to use this technology, when the other. I was involved in standardization of JAIN SLEE and am close to the standardization of SIP Servlets and feel motivated to contribute to the discussion.

Both technologies have proven well in at least the following three aspects: chosen in commercial deployments, performance and scalability and multi-protocol adaptation. They both allow carrier-grade performance, linear scalability and may be utilized with all kinds of networks. In JAIN SLEE the multi-protocol aspect is – to a certain extend – part of the architecture, for SIP Servlets a gateway approach is utilized allowing to interface with any networks and introducing service broker capabilities.

Before jumping into some details to position the candidates is to ask the question of technical and commercial implications of selecting the "wrong" technology. Selecting the "wrong" candidate for service execution will lead to messy architectures and most likely to a timely replacement of the installed platforms. Costs are high and competition might have better offerings to the public.
But comparing the impact of selecting the "wrong" execution platform with a non-organized approach to manage the execution platform, to connect the OSS[6] systems and BSS[7] systems to the execution environment and not structuring the organization properly will outweigh the investment in the execution part with a ratio of roughly around 1 to 5.

History

The standardization of JAIN SLEE started with JSR-22 in July 1999. The aim at this time was to become the Next Generation SLEE in telecommunication networks. JAIN SLEE was invented since JEE[8] at this time could not satisfy the demand for high throughput and low latency. JEE concentrated on long-running DB transactions and data modifications.

> *"The goal of SLEE is to replace proprietary SLEE's in the marketplace with a standardized application server for event processing. A primary focus of SLEE is to enable enterprise applications access to event driven networks in a standardized manner via standardized integration of the two environments. It is thought that integration of SLEE with J2EE will lead to converged services that blend for example, data and voice services."*[9]

The latest JAIN SLEE standard, JSR-240, finished at 14th July 2008.

The standardization of SIP Servlet started in April 2001. It was invented as an add-on to the JEE environment with the firm believe in JEE to evolve into a high-throughput environment due to improvements in hardware, JVM's and the JEE implementations as such.

> *"An important aspect of any communication infrastructure is programmability and the purpose of the SIP Servlet API is to standardize the platform for delivering SIP based services. The term platform is used here to include the Java API itself as well as standards covering the packaging and deployment of applications."*[10]

JSR-289, the latest SIP Servlet standard, aims to complete in July 2008 time frame.

Philosophy

Important when comparing things is to not compare apples with pears. Some people tend to compare JAIN SLEE (JSR-240) with SIP Servlets (JSR-289). Obviously, JAIN SLEE standardizes a whole application server environment including the programming model, the component model and the needed framework. SIP Servlets defines only the programming model and relies heavily on the component model and framework already defined by JEE. So, to make the overall comparison fair – to compare apples with apples – we should look at JAIN SLEE vs. SIP Servlet / JEE.[11]

JAIN SLEE

The below picture shows the architecture of a standard JAIN SLEE application server. It shows management capabilities, framework functions, component model and the resource adaptation layer.

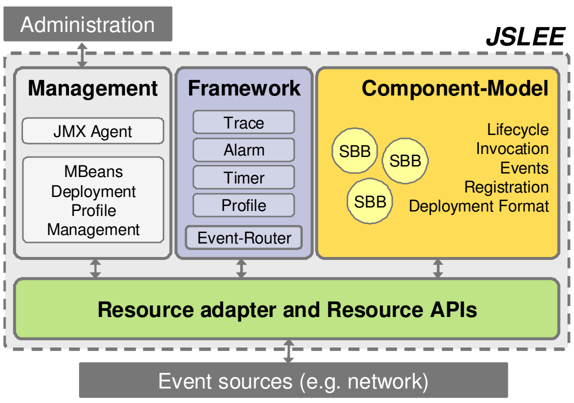

Figure 1 JAIN SLEE architecture overview

The components in JAIN SLEE are named SBB – service building block an allusion to IN systems. The component model and its composition mechanisms mimics the configurable state machines well-known from IN SCPs. In the model, everything is tailored to state transitions triggered by events emitted by the underlying networks. Composition of SBBs can be achieved through parent-child relations and prioritization of event delivery. The actual re-use of SBB components is limited through the specialisation needed to work as part of or as the state machine.
JAIN SLEE is a self-contained environment aiming for a complete SLEE environment, targets all networks per se and is ideally applied where IN systems need to be replaced by a modern system copying the fundamental approach of IN systems – the state machine. JAIN SLEE approaches network people with a solid understanding of the signalling network and aims to evolve towards software driven enterprise environments. It implements a bottom-up approach.
One of the real issues with JAIN SLEE is the complexity. The learning curve is steep. JAIN SLEE adds a complex execution environment on top of the complexity of signalling.

Available applications for JAIN SLEE include VPN, Virtual PBX, Mutli SIM, Personal Call Management, Missed Call Alerting. Commercial players in the market place include OpenCloud[12], jNETx[13] and Alcatel Lucent[14]. A Developer portal is offered by OpenCloud[15]. The "Mobicents" project[16] is an open-source implementation of the JAIN SLEE specification and gathers quite some community members.

SIP Servlet / JEE

The below shown architecture depicts the JEE environment with its component model, JEE services, management facilities, HTTP Servlets and the SIP Servlets.

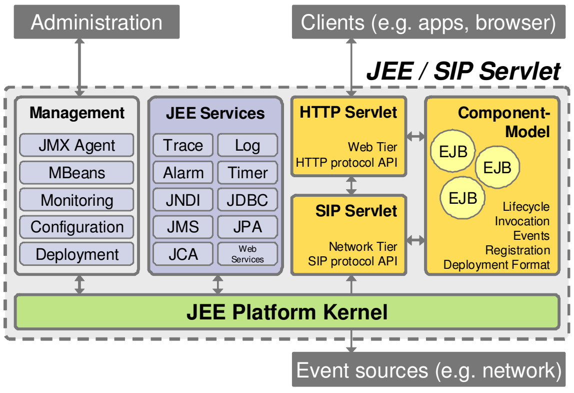

Figure 2 SIP Servlet / JEE architecture overview

JEE is a combination of 45 JSR's[17]. The SIP Servlet standard adds SIP signal protocol capabilities as another access channel to the JEE environment and formulates a programming model and an API to access SIP protocol messages – comparable to HTTP Servlets. The EJB component model allows a comparable programming approach as the one found in JAIN SLEE. However, the overall approach to enable IP and SIP based networks is not bound to a state machine model. It is closer to an enterprise-IT approach. The tight combination with the JEE environment allows an easy integration into well-known enterprise principles like e.g. SOA architectures.
However, the SIP Servlets based approach lacks support for other telecommunication protocols and needs gateways to communicate with non-IP based networks.
The overall idea of the JEE / SIP Servlet approach is to extend the broader enterprise platforms into the network, to extend the reach of the software driven enterprise environment into the network – a top-down approach.

The learning curve is flat for the SIP Servlets part and steep for a non-experienced JEE developer. The positive aspect is that the JEE community is mature and big. As said above, the real complexity is not in the programming model – it's hidden in the signalling.
Available applications for SIP Servlet include Centrex, Multimedia Ringback Tones, Mobile Advertisment, Number Translation, User Profile Based call routing, Video-Surveillance. Commercial players include IBM[18], Oracle[19] (ex-BEA), and Avaya[20] (ex-Ubiquity). Developer portals are offered at least by IBM[21], Oracle[22] and Avaya[23]. Open source projects include Sailfin[24], Mobicents SIP Servlet[25], SIPSEE[26] and WeSIP[27].

Multi-Protocol – How?

Well, multi-protocol support for an application server named "JEE/SIP Servlet" sounds a bit weird – solving it may even bring advantages. But let us start with looking at how JSLEE addresses multi-protocol support.

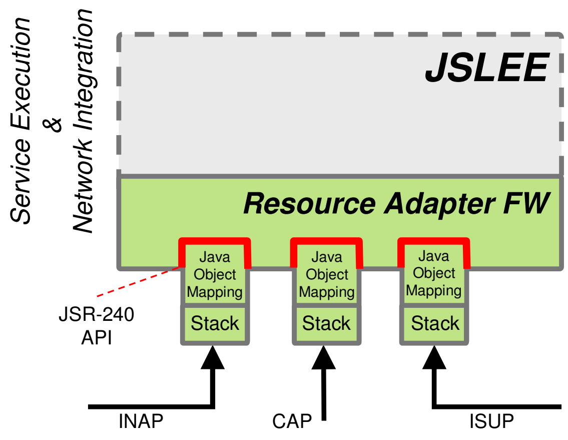

Figure 3 Multi-Protocol in JSLEE

JSLEE with its "Resource Adapter Framework" being specified in JSR-240 is per definition not bound to any protocol. A resource adapter contains the protocol stack and some mapping code translating the stack signals into Java objects. These objects are handed over to the event router for further processing in the JSLEE runtime. SBB's invoke methods of the resource adapter to generate protocol specific signals via the wrapped stack.
JSR-240 specifies the framework API – the syntax; it does not specify the stack-to-java mapping – the semantics. This is important to differentiate since the stack-to-java mapping introduces a tight coupling between the resource adapter and the SBB's and may limit the portability of applications.
The biggest advantage of this approach is the en-container support for multiple protocols and the JSR-240 standardized framework API.
Disadvantageous is the tight coupling between protocol ←→ java object mapping ←→ application which may prevent easy SBB portability. Furthermore, protocol translation and termination point is located in one single execution container which mainly limits the area of application of JSLEE to IN-like SCP[28].

In JEE/SIP Servlet, multi-protocol support is only achievable by plugging some external gateways to the execution environment – e.g. via the JCA[29] or some other protocols.

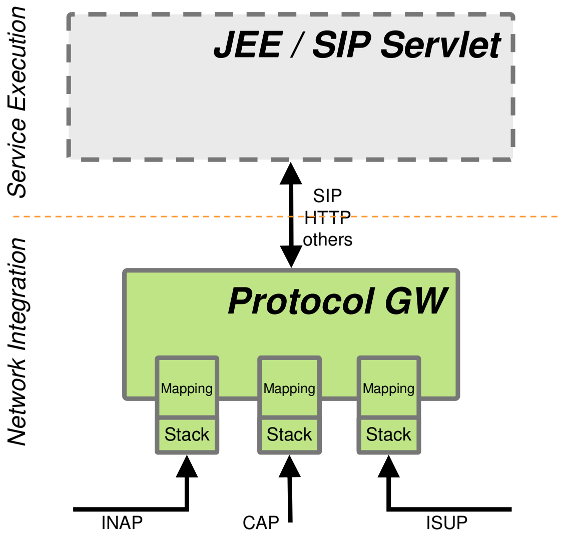

Figure 4 Multi-Protocol in JEE/SIP Servlet

Arriving network signals are terminated at a protocol stack and mapped via an internal vendor-specific mechanism into an intermediate representation of the network signal – ideally the SIP protocol. After some internal processing – e.g. functions like service brokering[30] – the intermediate representation is mapped towards e.g. SIP to signal the JEE/SIP Servlet application server. In those cases where the intermediate representation is already SIP based the overall encoding effort is quite slim.
The biggest advantage of this approach is the separation of concerns between network integration and service execution which allows specialized environments for both concerns. The network integration element furthermore allows the introduction of further functions like e.g. SCIM[31] or service broker functions. This is possible because the protocol

translation and termination point are not located in the same environment.
However, this approach increases the overall complexity with the need of having "yet another box" in the network and the flavour of an entirely proprietary solution.

Adding support for a new protocol is quite straight forward for both environments. In JSLEE it's the identification of a suitable stack vendor, the definition of stack-to-java mapping, the implementation of the wrapper covering the RA API and the previously defined mapping and finally the implementation of the SBB's relying on the just-defined protocol abstraction. In JEE/SIP Servlets, the steps are more limited however tightly bound to the vendor owning the protocol gateway. In JEE/SIP Servlets it's the purchase of the protocol support from the gateway vendor and the implementation of the application relying on the offered protocol support.

Component Composition

The component model of JSLEE and JEE have common roots since the SBB model is based on the principles of the EJB model. It is adapted to suite better the nature of an entirely event-driven runtime environment. Some limitations were introduced – e.g. no remote interfaces – and some functions stripped – e.g. the transaction context of EJB's.

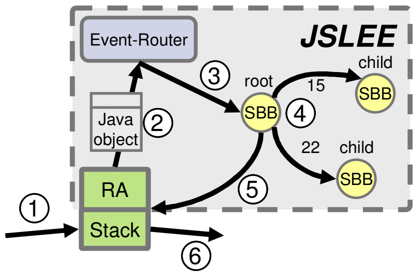

Figure 5 Component Composition and Event Processing in JSLEE

In the JSLEE model, a SBB is registered to receive certain types of events. These events are received by so called "root" SBB's. They represent the anchor of a composition tree containing some child SBB's. The child SBB's operate on the same signal – but execute different logic. The root SBB typically represents the "complete" application. A "CallForwarding" root SBB and a "CallBlocking" root SBB can act as child SBB's in an application "CallBlockingAndForwarding" represented by a root SBB. The sequencing of child SBB's is achieved via weighed connections between the root SBB and child SBB's. To communicate between the SBB's local interfaces are utilized. The component model in JEE/SIP Servlet is the EJB. EJB's are a well-known and controversial discussed concept. Nevertheless, the composition model allows re-use of existing functionality of various granularity being encapsulated in EJB's. Re-use can be configured during deploy-time via <ejb-ref> sections in the deployment descriptors. The EJB model is designed for distribution and allows local and remote interfaces to be exposed. The EJB-to-WebService mapping is part of the JEE standards procedures. The beauty of the JEE/SIP Servlet model is that the EJB component model is an optional part for heavily complex applications. Easier applications with less requirements for re-use could be implemented in SIP Servlets as the composition entity.

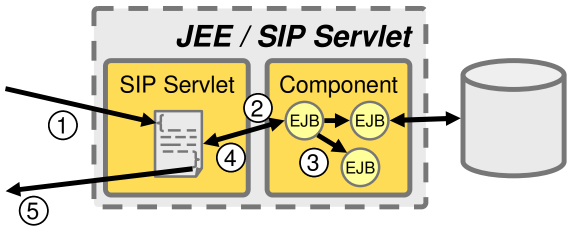

Figure 6 Component Composition and Event Processing in JEE/SIP Servlet

Event Processing

As depicted in Figure 5 in the JSLEE environment, a signal is processed by the associated network protocol stack (1). The stack hands over the information to the RA which maps the received information into the Java object representation. The Java object is then handed over to the event router (2). The event router notifies the subscribed root SBB's (3) and triggers off the event processing. Root SBB's distribute the received signal to their associated child SBB's for further processing. The sequence of notification follows the priorities assigned to the various child SBB's (4). After completing the processing of business logic in the JSLEE components the SBB invokes methods offered by the resource adapter (5). These methods trigger off the stack to create network protocol signals (6).

Figure 6 shows the JEE/SIP Servlet application environment and how the event is processed. The incoming SIP message is processed by the SIP stack and handed over to the SIP Servlet runtime environment (1). In less complex applications, the entity of an SIP Servlet might be enough to process the signalling. In these cases, the SIP Servlet contains all business logic and releases a network signal via the SIP stack (5) and skips the steps (2) to (4). In more complex scenarios, the SIP Servlet reaches out to business logic encoded in one or more EJB's (2). The EJB's use and re-use each others by simple references (3). External systems might be used by the means of the JEE runtime environment – databases might be queried via JDBC. The EJB processing finishes and returns an answer to the SIP Servlet (4). The SIP Servlet releases a SIP message back to the underlying SIP network (5). For the JEE environment the SIP Servlet container is – like the HTTP Servlet container – only another "access channel" to trigger processing in the component model.

Conclusion

So, who's the winner? Well, both technologies have friends and enemies. Most of the discussions about "either … or …" have the touch of religion – seeking for absolute answers.
In fact, multiple network operators around the world have both technologies installed in their networks. When it comes to commercial deployments, the area of application and how well the various technologies fit into the overall architecture is more important than "technological superiority".
It is of importance to choose the right technology for service execution, however, it is even more important to setup the overall organization and aligning the existing business processes to allow the creation and execution of these services than selecting the right execution technology for them.

For questions and comments please contact me via michael@maretzke.com.

[1] JAIN SLEE - "Java API's for Intelligent Networks" (JAIN) "Service Logic Execution Environment" (SLEE)
[2] JAIN SLEE 1.0 specified in JSR-22 see http://www.jcp.org/en/jsr/detail?id=22
[3] JAIN SLEE 1.1 specified in JSR-240 see http://www.jcp.org/en/jsr/detail?id=240
[4] SIP Servlet 1.0 specified in JSR-116 see http://www.jcp.org/en/jsr/detail?id=116
[5] SIP Servlet 1.1 specified in JSR-289 see http://www.jcp.org/en/jsr/detail?id=289
[6] OSS - Operation Support Systems
[7] BSS - Business Support Systems
[8] JEE – "Java Enterprise Edition" (formerly known as J2EE – "Java 2 Enterprise Edition")
[9] JSR-240 specification, Public Draft Review (PDR), page 4
[10] JSR-289 specification, Overview
[11] From a technical point of view, both technologies – SIP Servlet and JAIN SLEE – can be executed as stand-alone application servers or combined with a JEE environment. The JEE environment is usually utilized to integrate with backend systems and provide WebService exposure. In the case of SIP Servlet / JEE, the JEE environment provides the missing component model and framework functions.
[12] http://www.opencloud.com/
[13] http://www.jnetx.com
[14] Alcatel-Lucent Application Server http://www.alcatel-lucent.com/wps/portal/products/detail?LMSG_CABINET=Solution_Product_Catalog&LMSG_CONTENT_FILE=Products/Product_Detail_000102.xml&LMSG_PARENT=#tabAnchor4
[15] https://developer.opencloud.com/devportal/display/OCDEV/Home/
[16] http://www.mobicents.org
[17] JEE standards: http://jcp.org/en/jsr/tech?listBy=3&listByType=platform
[18] http://www-306.ibm.com/software/webservers/appserv/was/features/?S_TACT=105AGX10 &S_CMP=LP
[19] http://www.oracle.com/products/middleware/service-delivery-platform/docs/ communications-converged-application-server-datasheet.pdf,
[20] http://www.avaya.com/master-usa/en-us/resource/assets/factsheet/FINAL%20SIP%20AS%20DSHEET%20092007%20LB3551.pdf
[21] IBM's develeoper network http://www.ibm.com/developerworks/
[22] Oracle's developer network http://www.oracle.com/technology/index.html
[23] Avaya's developer network https://udn.devconnectprogram.com/
[24] https://sailfin.dev.java.net/
[25] https://sip-servlets.dev.java.net/
[26] http://www.fokus.fraunhofer.de/en/fokus_testbeds/open_ims_playground/components/sipsee/index.html
[27] http://www.wesip.com/
[28] SCP – Service Control Point
[29] JCA – Java Connector Architecture – see http://java.sun.com/j2ee/connector/
[30] A „service broker" is especially found in IN networks where a specialized entity needs to orchestrate IN applications and corelate the produced answers by the various SCP's.
[31] SCIM – Service Capability Interaction Manager as described in 3GPP TS 23.002 "*The application server may contain "service capability interaction manager" (SCIM) functionality and other application servers. The SCIM functionality is an application which performs the role of interaction management.*"